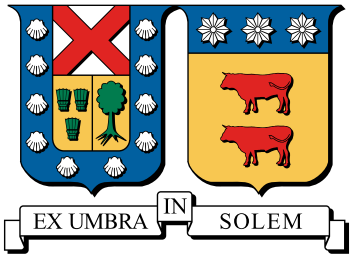
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAISO-CHILE



“Nueva interfaz para la base de datos de publicaciones de electrónica”

STEFANO EMMANUEL AGUILERA HUMENEY

MEMORIA DE TITULACIÓN PARA OPTAR AL TITULO DE INGENIERO CIVIL

ELECTRÓNICO MENCIÓN COMPUTADORES

PROFESOR GUÍA: Juan I. Yuz Eissmann

PROFESORES CORREFERENTE: Rudy Malonnek Wetzel

marzo - 2022

Nueva interfaz para la base de datos de publicaciones de electrónica

Stefano Aguilera

Memoria para optar al Título de Ingeniero Civil Electrónico, Mención en Computadores.

Universidad Técnica Federico Santa María

Profesor Guía: Juan I. Yuz Eissmann

Marzo - 2022

Resumen

Para las instituciones de educación superior y sus unidades académicas el registro de actividad y productividad científica se ha vuelto fundamental. Por esto es importante tener una interfaz para la base de datos de publicaciones y proyectos del departamento de electrónica, que sea simple, rápida de usar y se integre sin ningún problema a los servicios de otras entidades, como Web of Science.

En la presente memoria se detalla el diseño, desarrollo e implementación de una nueva interfaz para la base de datos de publicaciones del departamento de electrónica. La interfaz entrega una manera simple de visualizar la productividad individual y grupal de los miembros del departamento, como también poder actualizar la información desde otros repositorios de publicaciones. Por otro lado, se incluyen los proyectos de investigación y desarrollo de los miembros del departamento para facilitar el trabajo.

Para la realización de la nueva interfaz se decide mantener algunos aspectos de la antigua, como la base de datos en MySQL, mientras que en el lenguaje del servidor y cliente se decide usar Node.js y Svelte respectivamente, los cuales entregan sus propios beneficios y desafíos.

Palabras claves: Interfaz, web, base de datos, Node.js, svelte

New interface for the publications database of the electronic engineering department.

Stefano Aguilera

Final Project report towards the fulfillment of the Electronics Engineer, Minor in Computers degree.

Universidad Técnica Federico Santa María

Advisor: Juan I. Yuz Eissmann

March - 2022

Abstract

Nowadays, the registry of scientific productivity and activity is fundamental for higher education institutions. For this, it is important to have an interface for the publications and projects database of the electronic engineering department, that is simple, easy to use and which integrates to services from other entities, such as Web of Science.

In this report, the design, development and implementation of a new interface for the electronics department database is presented. This interface provides a simpler way to view the individual and group productivity of its members, as well as the ability to update the database information with another repositories. On the other hand, it also considers in the database the research and development projects of the department members to facilitate their work.

For the development of this new interface some characteristics of the current platform were kept (e.g., the database in MySQL), while others like the server and client language were changed to use Node.js and Svelte, which give their own benefits and challenges.

Keywords: Interface, web, database, Node.js, svelte

Glosario

Back end: Parte del software ubicado en el servidor que procesa las acciones entregadas por el usuario mediante el front end.

Front end: Parte del software con la cual interactúa el usuario de forma gráfica, siendo responsable de enviar información al back end y mostrar datos de este al usuario final.

DOI (Digital Object Identifier): Identificador único de objeto digital usada para publicaciones que funciona como vínculo al origen.[5]

ISSN (International Standard Serial Number): Identificador único de medios impresos y digitales.[11]

HTML (HyperText Markup Language): Lenguaje de marcado de hipertexto con el cual se define la estructura básica de las páginas web. [16]

CSS (Cascading Style Sheets): Lenguaje de diseño gráfico con el cual se define el estilo de las páginas y componentes. [8]

Cookie: Trozo de información creado por el servidor y que se almacena en el cliente de manera de conocer la actividad previa. [2]

WoS (Web of Science): Servicio web de información científica, guarda y entrega información de publicaciones, revistas y autores. (Pertenece a Clarivate Analytics) [30]

URL (Uniform Resource Locators): Dirección única de un recurso web. [3]

GIT: Software de control de versiones que lleva registro de los cambios que tengan los archivos y permite que estos sean modificados por un grupo de personas, haciéndose cargo de los conflictos en los archivos que puedan tener.

Visual Studio Code: Entorno de desarrollo integrado que entrega las herramientas para el desarrollo de software.

Índice general

Índice de figuras

Capítulo 1

Introducción

El Departamento de Electrónica de la Universidad Técnica Federico Santa María en la actualidad cuenta con una interfaz para la base de datos de publicaciones, la cual permite subir y visualizar publicaciones, pero carece de indicadores bibliométricos y factores de impacto de revistas, y carece de integración con otras plataformas ya existentes.

Es por esto que se busca realizar una nueva interfaz que solucione los problemas de la anterior, entregando así una página que facilite el trabajo a los académicos, permitiendo la búsqueda de publicaciones e información de manera simple y rápida, así como también que puedan actualizar sus curriculums y extraer estadísticas. Si bien ya existen otras interfaces, se decide crear una nueva para cumplir con los requerimientos usuales de la universidad, como poder incluir los proyectos de investigación, y para tener mayor libertad a la hora de decidir las características que esta tenga.

Dicho todo esto, se definen los siguientes objetivos que cumple el proyecto:

- Poder consultar y visualizar de manera simple la productividad individual de los miembros del departamento con el fin de facilitar el análisis o actualizar sus curriculums.
- Poder consultar y visualizar de manera simple la productividad grupal del departamento y de sus miembros, por ejemplo, para procesos de acreditación.
- La interfaz debe poder actualizarse utilizando información de otros repositorios de publicaciones de manera simple y expedita, con el fin de reducir el tiempo necesario

para agregar publicaciones. Por esto se busca que se integre con interfaces externas, tales como Crossref, Web Of Science, ORCID y Google Scholar.

- Incorporar a la base de datos los proyectos de investigación y desarrollo del departamento. Dado que también son trabajos de los miembros del departamento, se agregan para facilitar la actualización de curriculums y que estos se puedan relacionar a sus miembros.

Para esto se analiza el estado del arte con respecto a plataformas de bases de datos y sus interfaces para publicaciones académicas y se seleccionan las herramientas con las cuales se logran los objetivos, que se detallan en el siguiente capítulo.

Capítulo 2

Estado del arte

El objetivo principal es actualizar la interfaz actual de publicaciones del departamento. Para esto, se considera la siguiente estructura: Front end, back end y base de datos. Ilustrado en la Figura 2.1. A continuación, se muestran las distintas alternativas existentes para el desarrollo.

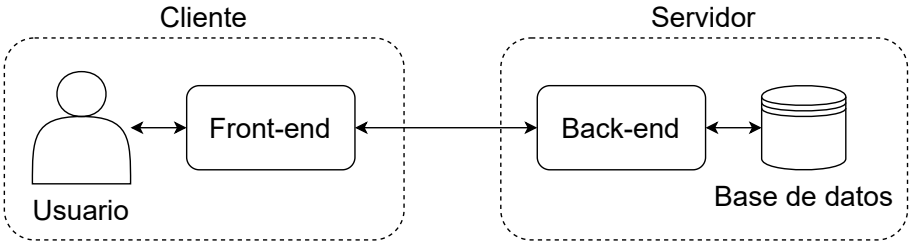


Figura 2.1: Modelo del sistema

2.1. Interfaces existentes

- **Google Scholar** [10]: Servicio de búsqueda de publicaciones. Se caracteriza por la facilidad de búsqueda y acceso a un gran catálogo de distintas instituciones. Además, proporciona al autor un perfil publicado y permite ver quien está citando sus publicaciones.

- **ORCID** [18]: Sistema de identificación para investigadores, cuyo fin es estandarizar la identificación de investigadores para todo tipo de aplicación o sistema.

- **Scopus** [25]: Base de datos de literatura científica proporcionada por Elsevier. Contiene libros, actas de conferencias y revistas científicas. Para facilitar el acceso a estos contenidos, Scopus proporciona herramientas para acceder, analizar y visualizar la información.

- **Mendeley** [13]: Gestor de referencias desarrollado por Elsevier. Ayuda a guardar, organizar y compartir referencias bibliográficas e información de investigación. Está más enfocado a los investigadores ya que su objetivo es facilitar la búsqueda de información.

- **Pure** [22]: Sistema de gestión de información (proporcionado por Elsevier). A diferencia de Google Scholar, Pure proporciona toda la interfaz para la gestión de todo el entorno de investigación (entregando reportes de las investigaciones, generando perfiles de investigadores y combinando los sistemas internos con datos de fuentes externas en una sola plataforma). Dado que es desarrollado por Elsevier, proporciona una integración sin problemas con Scopus y PlumAnalytics.

- **DSpace** [7]: Aplicación de código abierto para la creación de repositorios digitales. Si bien no tiene un enfoque directo al mundo académico como los servicios anteriores, si tiene todas las herramientas necesarias para ser usado en este. Se caracteriza por ser código abierto, su agilidad y flexibilidad para la implementación.

- **VIVO** [28]: Aplicación de código abierto que entrega todo un entorno para la gestión de información académica. En sus funcionalidades es similar a Pure, entregando la capacidad de buscar publicaciones, obtener estadísticas de impacto y facilita la generación de perfiles de investigador. Tiene integración con ORCID y puede obtener datos de Web Of Science y Elsevier.

- **Publons** [21]: Servicio de seguimiento de impacto de investigación desarrollado por Clarivate Analytics. Proporciona un sistema con el cual los investigadores pueden hacer seguimiento de sus publicaciones, la métrica de citaciones y revisiones de pares en un solo perfil.

2.2. Alternativas de Solución

En esta sección se describen las distintas alternativas disponibles para el desarrollo de la interfaz dividido en las 3 secciones de desarrollo del software.

2.2.1. Back end

1. **PHP** [19]: Lenguaje de programación de código abierto desarrollado por PHP Group. Se caracteriza por ser especialmente adecuado para el desarrollo web y puede ser incrustado en las páginas HTML.

   - **Fortalezas**:

     - Permite ser incrustado en las páginas HTML para entregar páginas de forma dinámica.
     - Tiene una gran comunidad ya establecida, lo que entrega una gran variedad de paquetes para facilitar el desarrollo.
     - Buen soporte para el procesamiento y manejo de texto.

   - **Debilidades**:

     - No comparte recursos entre procesos. Lo que se hace un problema a la hora de escalar el sistema por el gran uso de recursos.
     - No soporta de forma nativa la programación asincrónica. Esto significa que, si un proceso requiere la lectura de una base de datos o un archivo, este se pausara hasta recibir respuesta.

2. **Node.js** [17]: Es un entorno de ejecución de Javascript basado en el motor Chromium de Google. Su objetivo es poder desarrollar aplicaciones JavaScript que corran en el servidor e interactúen con el cliente.

   - **Fortalezas**:

     - Permite la instalación de paquetes, los cuales entregan más funcionalidades y facilitan el desarrollo del código.

- Al estar basado en JavaScript, la curva de aprendizaje es rápida ya que es el mismo lenguaje que se usa en el desarrollo del front-end.
- Permite ejecución en base a eventos, lo cual permite que se acceda a los datos en tiempo real.

▪ **Debilidades**:

- Al tener tipos de datos dinámicos, se hace difícil el reconocimiento de estos.
- Dado que se basa en gran parte en eventos, se hace complejo seguir la lógica de ejecución del programa.

3. **Go** [9]: Es un lenguaje de programación concurrente y compilado desarrollado por Google. Se inspira en la sintaxis de C e intenta ser dinámico como Python y con el rendimiento de C.

▪ **Fortalezas**:

- Dado que es simple y dogmático, es fácil de aprender y códigos escritos en Go con la misma funcionalidad tienden a verse iguales.
- Tiene soporte nativo para concurrencia en forma de rutinas y canales, lo cual permite ejecutar código en paralelo.
- Tiempos de compilación mucho más rápido que otros lenguajes en la misma línea (es decir, que requieren compilación)

▪ **Debilidades**:

- Dado que es simple, carece de funcionalidades más avanzadas tales como tipos de datos algebraicos y coincidencia de patrones.
- Es muy orientado a la programación imperativa, con lo que la programación funcional no es muy práctica.

4. **Python** [23]: Lenguaje de programación interpretado administrado por la Python Software Foundation. Se caracteriza por ser multiparadigma y fácil de leer y aprender.

▪ **Fortalezas**:

- Como se dijo anteriormente, es facil de aprender y leer. Siendo explícito a la hora del desarrollo.
- Presenta una gran comunidad ya establecida, entregando una gran variedad de librerías para facilitar el desarrollo.
- Solamente se necesita tener el intérprete de Python para su ejecución, el cual se encuentra disponible para una gran variedad de entornos.

- **Debilidades**:
  - Al ser interpretado, es más lento que los lenguajes compilados.
  - Si bien presenta un colector de basura que limpia la memoria que no se usa, no es muy eficiente, llegando a ocupar una gran cantidad de memoria disponible.

5. **Java** [12]: Lenguaje de programación orientado a objetos desarrollado por Oracle Corporation. Se caracteriza por permitir que el código pueda ser ejecutado en cualquier plataforma, sin ser compilado nuevamente.

   - **Fortalezas**:
     - Permite la ejecución del código en cualquier sistema operativo.
     - Presenta un colector de basura que se encarga de liberar memoria durante la ejecución del programa.
   - **Debilidades**:
     - Dado que virtualiza el código para que se pueda ejecutar en cualquier plataforma, se reduce el rendimiento del programa.
     - Como está orientado a objetos, la curva de aprendizaje puede ser complejo para aquellos que no estén familiarizados.

6. **C#** [4]: Lenguaje orientado a objetos desarrollado por Microsoft como parte de la iniciativa .NET y posteriormente fue aprobado como estándar internacional por ECMA e ISO.

   - **Fortalezas**:

- Soporta tanto programación orientada a objetos como también programación funcional, y presenta una gran variedad de funcionalidades a la hora de resolver un problema.
- Dado que es desarrollado por Microsoft, presenta uno de los mejores entornos de creación con Visual Studio, el cual además tiene IntelliSense que permite relacionar todo el código del programa en el proceso de desarrollo.

- **Debilidades**:
  - Dado que es desarrollado por Microsoft, no presenta el mismo rendimiento y soporte a la hora de ejecutarlo en otros sistemas operativos como OSX y Linux
  - Comparado con otros lenguajes, carece de ciertas funcionalidades que estos tienen.

2.2.2. Front end

- **Angular** [1]: Framework para aplicaciones web desarrollado por Google. Desarrollado en TypeScript, es utilizado para desarrollar aplicaciones web de una sola página y altamente adaptable a los distintos tipos de plataformas.
  - **Fortalezas**:
    - Presenta una documentación bien clara y un gran soporte de la comunidad.
    - Se estructura de manera modular lo que permite un desarrollo altamente escalable de la interfaz
    - Al tener una arquitectura de vista-controlador, se logra separar la lógica de la aplicación con la interfaz del usuario.
  - **Debilidades**:
    - Puede ser engorroso de entender debido a la gran cantidad de palabras que se utilizan.
    - Gran uso de recursos, lo cual puede ser un problema en dispositivos antiguos.

- **React** [24]: Biblioteca de Javascript para construir interfaces de usuario desarrollado por Facebook. Su objetivo es facilitar el desarrollo de aplicaciones que usan datos cambiantes.

  - **Fortalezas**:
    - Facilita el desarrollo de aplicaciones dinámicas
    - Tiene una buena documentación y un gran soporte de la comunidad
    - Al actualizar alguna parte de la interfaz, solo se actualiza ese apartado sin afectar el renderizado de toda la aplicación.
  - **Debilidades**:
    - Al usar JSX, que es una extensión de la sintaxis de JavaScript para usar HTML en este, el código puede llegar a ser bastante complejo de entender.
    - Al ser una librería, no tiene un modelo estandarizado para la estructura de la aplicación como el de Angular, siendo responsabilidad del usuario la administración de la aplicación.

- **Vue** [29]: Framework para aplicaciones web creado con la idea de tener las mismas funcionalidades que Angular pero mucho más liviano.

  - **Fortalezas**:
    - Pequeño tamaño, Al pesar solo 18KB, es rápido de instalar.
    - Fácil de aprender, presenta una sintaxis bastante similar a Javascript, por lo que no requiere mucho aprendizaje.
    - Buena documentación.
  - **Debilidades**:
    - Gran uso de recursos, lo cual puede ser un problema en dispositivos antiguos.
    - Al ser relativamente nuevo, no hay suficientes extensiones que puedan entregar la funcionalidad necesaria que se busca.

- **Svelte** [27]: Framework para aplicaciones web relativamente nuevo caracterizado por

tener un compilador que integra los distintos lenguajes del desarrollo web (HTML, CSS y JavaScript).

- **Fortalezas**:
  - Facilidad de lectura del código al tener un compilador que los separa de manera simple de leer.
  - Mucho más eficiente que las otras alternativas debido al compilador.
  - Si se usa Node.js, se puede usar Sveltekit para simplificar aún mas el trabajo.
- **Debilidades**:
  - Debido a su desarrollo reciente, no es muy estable.
  - Por lo anterior tampoco se tiene una comunidad establecida.

2.2.3. Base de datos

- **MongoDB** [14]: Sistema de gestión de bases de datos no relacional desarrollado por MongoDB. Almacena los datos en formato JSON.
  - **Fortalezas**:
    - Fácil lectura al estar almacenado en JSON
    - Gran escalabilidad en su servicio.
    - Al no ser relacional, presenta una gran flexibilidad en los datos a almacenar.
    - Buena capacidad de recuperarse ante fallas.
  - **Debilidades**:
    - No se recomienda para datos ordenados y relacionales
    - No se puede realizar transacciones entre distintos documentos.
    - No es recomendada para aplicaciones con transacciones complejas
    - Es necesario pagar licencia para uso comercial.
- **MySQL** [15]: Sistema de gestión de bases de datos relacional, de código abierto y desarrollada por Oracle.

- **Fortalezas**:
  - Gran soporte de la comunidad, debido a su larga trayectoria.
  - Entrega una gran cantidad de herramientas y funciones para facilitar el manejo y configuración de la base de datos.
  - Gran velocidad a la hora de entrega y manejo de datos.
  - Posee una herramienta gráfica para el diseño y administración de bases de datos.
- **Debilidades**:
  - Si se desarrolla un producto comercial es necesario pagar por la licencia de MySQL.
  - Al tener los tipos de datos definidos no permite una gran flexibilidad a la hora de guardar datos.

- **PostgreSQL** [20]: Sistema de gestión de bases de datos relacional orientado a objetos y de código libre desarrollado por PostgreSQL Development Group.
  - **Fortalezas**:
    - Posee una herramienta gráfica para el diseño y administración de bases de datos.
    - Buena escalabilidad, siendo capaz de ajustar los recursos disponibles.
    - No requiere pago de licencias.
  - **Debilidades**:
    - La sintaxis no es intuitiva, siendo difícil de aprender.
    - En bases de datos pequeñas no es óptimo.

2.3. Alternativa Seleccionada

Las alternativas seleccionadas fueron elegidas en base a la estabilidad y seguridad, la eficiencia que tienen para la resolución de problemas, sus curvas de aprendizaje y la comunidad que tienen detrás para la simplificación de problemas. Un criterio relevante que no se puede dejar de lado es el sistema que usa el departamento, y por otro lado la base de datos que ya se viene usando. Por lo que se seleccionan las siguientes alternativas:

- Base de datos - **MySQL** [15] Base de datos relacional bastante estable y con una comunidad ya establecida. Se decidió mantener la alternativa que ya se venía usando de antes, esto debido a su estabilidad y para evitar que se complique la transferencia de información.

- Back end - **Node.js**[17] Destaca por su comunidad ya establecida que presenta una gran cantidad de librerías para simplificar el trabajo. Por otro lado, comparte el lenguaje JavaScript. Dado que en el front end se usa Svelte, se utiliza en conjunto con SvelteKit, el cual entrega una sinergia que permite simplificar el trabajo.

- Front end - **Svelte**[27]: Destaca por su simplicidad y eficiencia. Se caracteriza por integrar de manera simple y fácil de leer los distintos lenguajes del desarrollo web (HTML, CSS y JavaScript), lo cual permite que se pueda trabajar de manera más rápida. Por otro lado, presenta una integración con el back end en Node.js permitiendo que el desarrollo conjunto se simplifique.

Capítulo 3

Diseño de interfaz

Para implementar una nueva interfaz para la base de datos de publicaciones es necesario en primer lugar tener en cuenta las consideraciones de diseño y los requisitos necesarios.

3.1. Requisitos de diseño de backend

Para el backend es necesario tener en cuenta los siguientes requisitos de diseño

1. Debe ser capaz de trabajar con la base de datos ya existente, la cual está hecha en MySQL.

2. Debe ser capaz de interactuar sin problemas con el frontend para recibir consultas y entregar la información de la base de datos.

3. Debe ser capaz de interactuar con otras bases de datos de plataformas ya existentes (Por ej. CrossRef) para facilitar el ingreso de información.

4. El código debe ser fácil de comprender y escalable.

3.2. Estructura preliminar Back End

Para el desarrollo del Back End es necesario tener en consideración la estructura bajo la cual se trabajará. Por esto, se debe tener en cuenta sus partes (que se ilustran en la figura 3.1).

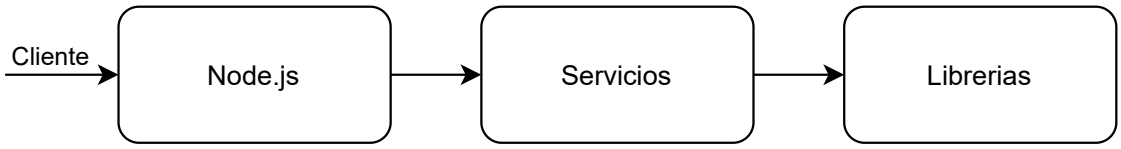


Figura 3.1: Estructura de backend

- **Node.js**: Entorno encargado del manejo de las conexiones HTTP tanto entrantes como salientes. Se usa junto al módulo SvelteKit con el fin de facilitar la interacción cliente-servidor.
- **Servicios**: Módulo encargado de realizar las operaciones necesarias para la entrega de la información solicitada por el cliente. Estos se encuentran divididos en múltiples archivos .ts (TypeScript).
- **Librerias**: Módulo encargado de la interacción con otras plataformas, sea esta la base de datos, o una plataforma externa (Como Crossref o ORCID).

Cabe destacar la separación entre la lógica del software y las librerías, esto con el fin de permitir escalabilidad del código y facilidad de mantención.

3.2.1. Control de acceso

Para controlar el acceso a las funcionalidades que modifiquen los datos, se usa una *whitelist* que permite el acceso únicamente a los usuarios que se encuentren en ella. Por otra parte, para verificar la identidad del usuario se usa la plataforma de pasaporte USM, que permite verificar la identidad del usuario como miembro del departamento de electrónica.

La modificación de la *whitelist* debe ser posible solo por la cuenta de administrador, la cual es única y su contraseña está definida en el archivo de entorno del servidor (`.env`)

3.3. Estructura de la Base de Datos

Con el fin de tener toda la información ordenada y de fácil acceso se utiliza el modelo detallado en la figura 3.2. En esta figura, se presentan las tablas para el guardado de publicaciones, autores, revistas, proyectos, usuarios, comentarios y agencias.

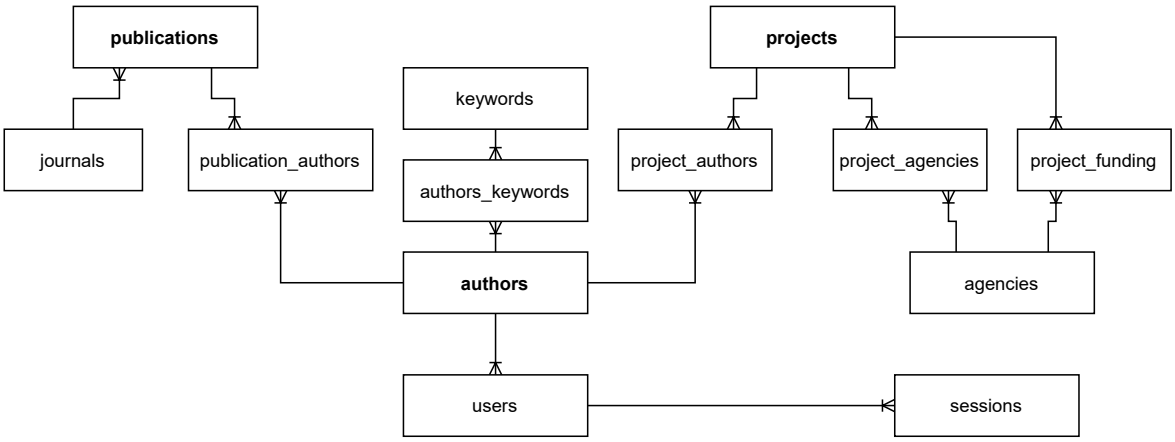


Figura 3.2: Modelo de base de datos

3.3.1. Tablas de publicaciones

- **publications**: Tabla en la cual se guarda toda la información general relativa a la publicación, siendo en su mayoría definida por el estandar BiBTEX.

    - **id**: ID única de publicación.
    - **title**: Titulo de publicación.
    - **entry_type**: Define el tipo de la publicación que en su mayoria estan definidos por el estandar BiBTeX.
    - **uploaded**: Fecha de subida de publicación a la base de datos.
    - **abstract**: Resumen del contenido de la publicación.
    - **id_journal**: ID de revista en que se publica.
    - **year,monthX,dayX**: Fecha de publicación o rango de conferencia.
    - **doi**: Codigo DOI asociado a la publicación.
    - **wosuid**: Codigo de WoS asociado a la publicación.
    - **booktitle**: Titulo de libro.
    - **volume**: Volumen de libro o revista.
    - **number**: Numero de libro o revista.
    - **pages**: Paginas relacionadas a la publicación.
    - **location**: Lugar de conferencia.
    - **url, url_access**: URL asociada a la publicación.
    - **pdf, pdf_access**: PDF asociado a la publicación.
    - **degree**: Grado o titulo asociado a la tesis o memoria publicada.
    - **note**: Notas en general.
    - **nousm**: Indica si la publicación no tiene relación a la universidad.
    - **toappear**: Indica si la publicación aun no está publicada.
    - **national_conference**: Indica si la publicación es de una conferencia nacional.

- **institution**: Institución asociada al grado/título de tesis/memoria.

- **publication_authors**: Dado que una publicación puede tener más de un autor, se define esta tabla con el fin de relacionar las publicaciones a la tabla de autores, agregando por medio de un campo para definir si es autor/editor de una publicación o el autor/supervisor/cosupervisor de una memoria/tesis, como también la posición en que debería aparecer el autor.

  - **publications_id**: ID de publicación a relacionar.
  - **authors_id**: ID de autor de publicación.
  - **type**: Define el tipo de la relación, siendo estos de autor, editor, supervisor o cosupervisor.
  - **place**: Lugar en que debe aparecer.

- **journals**: A diferencia de la interfaz anterior, se dio paso a guardar información relacionada a la revista en el cual se publica un artículo, incluyendo información como el factor de impacto y códigos ISSN asociados.

  - **id**: ID única de revista.
  - **title**: Titulo de revista.
  - **issnp**: Código ISSN de revista impresa.
  - **issne**: Código ISSN de revista electrónica.
  - **impact_factor**: Factor de impacto de revista.
  - **wos**: Bit que indica si se encuentra en Web Of Science.

3.3.2. Tablas de proyectos

- **projects**: Tabla en que se guarda toda la información general relativa a los proyectos, siendo por ejemplo su título, fechas de inicio y termino, fecha de financiamiento, tipo y código asociado.

  - **id**: ID única de proyectos.

- **title**: Titulo de proyecto.
- **start,end**: Fechas de inicio y termino de proyecto.
- **type**: Tipo de proyecto. Siendo estos entre investigación, innovación u otros.
- **code**: Código del proyecto.
- **note**: Notas del proyecto.
- **nousm**: Indica si no tiene relación con la universidad.

- **project_funding**: Dado que los proyectos pueden tener más de un financiamiento y una misma agencia puede entregar distintos fondos (Por ejemplo ANID con fondos FONDECYT y REDES), se define esta tabla con el monto y la agencia que financia.
  - **projects_id**: ID de proyecto asociado.
  - **agencies_id**: ID de agencia que financia.
  - **amount**: Cantidad entregada.
  - **type**: Tipo de financiamiento.
  - **date**: Fecha de financiamiento.
- **project_agencies**: Si bien un financiamiento pasa por una agencia, no siempre alguna agencia relacionada a un proyecto es financiera, es por esto que se deja en esta tabla aparte, con el fin de definir el tipo de relación que tenga.
  - **projects_id**: ID de proyecto asociado.
  - **agencies_id**: ID de agencia asociada.
  - **type**: Tipo de asociación. Siendo estos principal o asociada.
- **project_authors**: Define los autores relacionados con el proyecto, definiendo el rol que este tienen.
  - **projects_id**: ID de proyecto asociado.
  - **authors_id**: ID de autor asociado.
  - **role**: Rol de autor en el proyecto (Ej. Director/Principal investigador/Investigador).

- **agencies**: Tabla que contiene la información general de las agencias:
  - **ID**: ID único de agencia.
  - **title**: Nombre de la agencia.
  - **acronym**: Acrónimo de la agencia.

3.3.3. Tablas de autores

- **authors**: Tabla en que se guarda toda la información general relativa al autor, siendo por ejemplo sus nombres, asociación a la universidad, página web y códigos asociados de otras plataformas.
  - **ID**: ID única de autor.
  - **surname**: Apellido de autor.
  - **name**: Nombre de autor.
  - **usm**: Define si el autor está actualmente en el departamento, paso por este o no tiene relación alguna.
  - **homepage**: Pagina web del autor.
  - **orcid**: Codigo ORCID asociado al autor.
  - **researcherID**: Codigo researcherID asociado al autor.
  - **googleScholar**: Codigo googleScholar asociado al autor.
  - **scopusID**: Codigo asociado a la plataforma Scopus asociado al autor.
- **authors_keywords**: Tabla que relaciona palabras claves para la búsqueda del autor en cuestión.
  - **authors_id**: ID de autor.
  - **keywords_id**: ID de palabra clave asociada.

3.4. Estructura preliminar de Front End

Para diseñar el front end es necesario entender cómo funciona y trabaja la librería de Svelte. De su documentación se rescatan los siguientes puntos:

- **Formato de archivos**: Svelte trabaja con su propio formato de archivo (.svelte) donde se divide en 3 secciones dependiendo del código web con el que se trabaja (JavaScript, HTML y CSS) lo que facilita la lectura y comprensión del código. (Ilustrado en la figura 3.3)

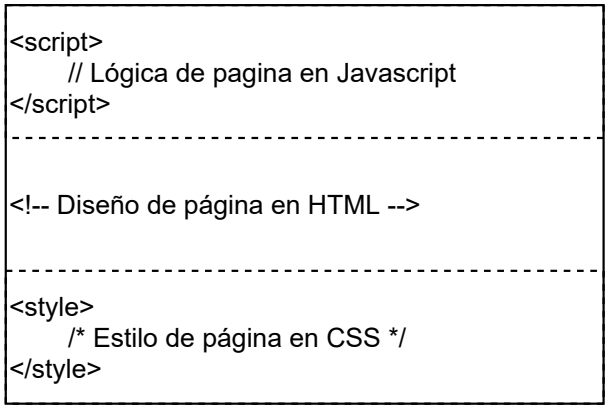


Figura 3.3: Estructura de archivo .svelte

- **Trabajo con componentes**: Los componentes son archivos con su propio código web que pueden ser utilizados en las páginas, con esto se evita escribir el mismo código en distintos archivos y permite mantener ordenado el proyecto. (Ilustrado en la figura 3.4)

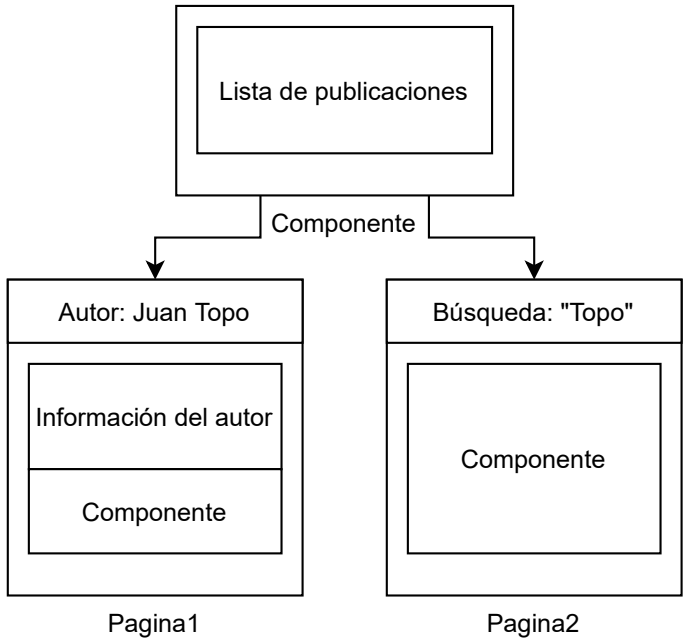


Figura 3.4: Ejemplo de componentes

Por otro lado, dado que se trabaja con Node.js y svelteKit hay que considerar las rutas a las cuales tiene acceso el cliente, definidas de la siguiente forma:

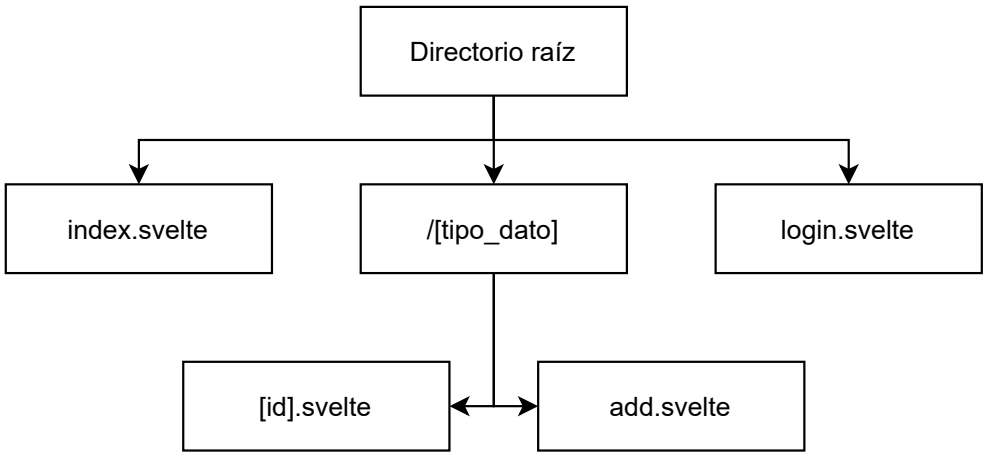


Figura 3.5: Rutas de cliente

Uno de los beneficios de usar la librería svelteKit es que permite el uso de parámetros dinámicos definidos en [ ], con lo que una misma página puede servir a diferentes consultas. Por ejemplo, si un cliente quiere ingresar a la página */publications/453*, este recibirá el contenido de la página */[tipo_dato]/[id].svelte*

Además, dado que las páginas suelen tener elementos que deberían estar presentes en todas las páginas (como las barras de navegación) se permite el uso del archivo *__layout.svelte*.

3.4.1. Página de inicio

Para la realización de la página de inicio a diferencia de la interfaz anterior se decidió ir por un estilo minimalista (siguiendo los pasos de otros buscadores web) con el fin de que sea más intuitiva (Ilustrado en la figura 3.6). Para esto se diseña la pagina de inicio con un campo de búsqueda, filtros y la lista de publicaciones que aparece según los resultados.

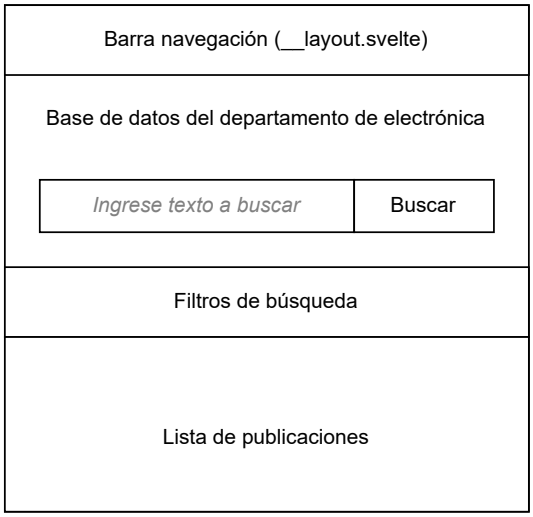


Figura 3.6: Borrador de index.svelte

3.4.2. Página de inicio de sesión

Con el fin de permitir el acceso a los componentes de edición es necesario el inicio de sesión, para esto se diseña una interfaz (Definida en la figura 3.7) que permite el acceso de manera simple.

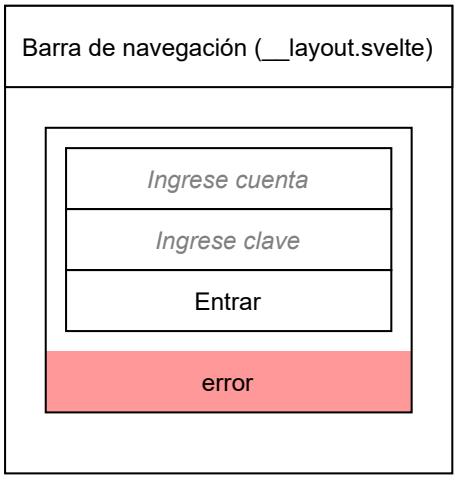


Figura 3.7: Borrador de login.svelte

3.4.3. Página de visualización de datos

La visualización de datos está dividida entre distintas paginas dependiendo del tipo de información que se quiera visualizar, pero todos siguen un formato general común con el fin de mantener la estética de la interfaz. Para esto, se presenta con un título, seguido con los datos referentes a lo que se consulta y por último dependiendo del tipo se muestra o no una lista de publicaciones relacionadas. (Ilustrado en la figura 3.8)

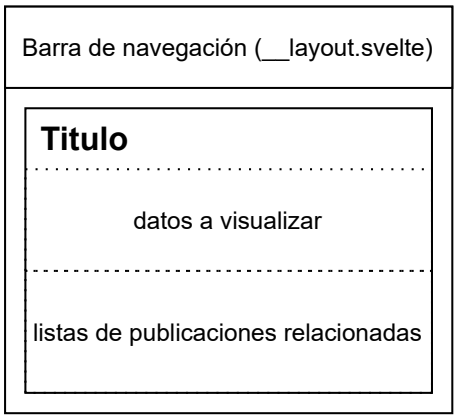


Figura 3.8: Borrador de [id].svelte

3.4.4. Página de agregado y edición de datos

De igual manera que la página de visualización, están divididas dependiendo de la información, pero también se sigue un formato general. Cabe destacar que esta página se deja aparte para prevenir que se edite información por un cliente no autorizado.

Para esto, se presentan todos los campos que tiene la información en la base de datos para modificar, como se ilustra en la figura 3.9

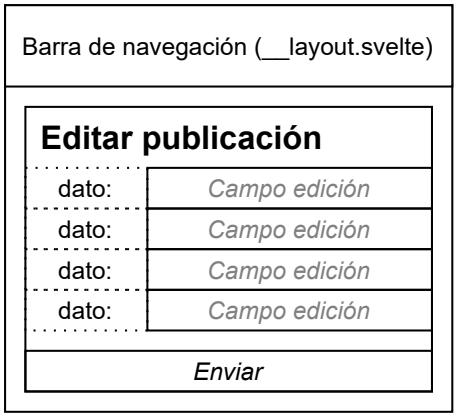


Figura 3.9: Borrador de add.svelte

3.4.5. Páginas de administración

Para la modificación de funcionalidades de administración (como la *whitelist*) se tiene una página de ingreso especial muy similar a la de inicio de sesión, la cual permite el acceso a la pagina de modificación de la whitelist que permite agregar o quitar el acceso de los usuarios.

Capítulo 4

Desarrollo de interfaz

4.1. Entorno de desarrollo

Para el desarrollo de la interfaz y simplificar el futuro despliegue de esta, se decide trabajar en un computador personal bajo GIT, aplicación que permite desplegar de manera fácil la interfaz juntos con los cambios que vengan a futuro. Para esto se usa BitBucket, servicio que permite tener repositorios GIT privados en la web sin necesidad de pago.

Para hacer uso de esto, es necesario crear un repositorio en la página, tener instalado GIT en la computadora y subir la interfaz al repositorio web. Para esto se usa el comando ’ git clone’ para clonar el repositorio externo en la computadora personal. Posteriormente todo cambio que se quiera subir al repositorio GIT de Bitbucket se sube con los comandos:

```
1 git add <archivo a subir>
2 git commit -m <comentario de los cambios a subir>
3 git push
```

Listing 4.1: Comandos para subir cambios al repositorio

También se puede usar la interfaz gráfica que entrega Visual Studio Code, la cual permite subir cambios de los archivos de manera fácil y simple.

Para el desarrollo de la interfaz se usa el subsistema de Windows para Linux (WSL2) para simular el entorno que generalmente se encuentran en los servidores, Además de que se integra fácilmente con Visual Studio Code. Se instalan los paquetes necesarios que en su

mayoría son los mismos que se usarán en el servidor y se procede a preparar el entorno de trabajo con los siguientes comandos que dejan en línea un servidor en el computador personal para el desarrollo de la interfaz:

```
1 npm init svelte@next <nombre_aplicacion>
2 cd <nombre_aplicacion>
3 npm install
4 npm run dev
```

Listing 4.2: Comandos para preparar el entorno de trabajo

4.2. Flujo de consultas en el servidor

Para comprender el funcionamiento de los módulos del servidor es necesario entender primero el flujo de las consultas en el servidor Node.js y svelteKit, en donde el servidor de Node.js recibe la consulta para posteriormente ser preprocesada por el archivo hooks.ts, dependiendo del resultado la consulta pasa por el enrutador de svelteKit, el cual decide el módulo al que llamar (Siendo este una página o un script en el servidor). Este flujo queda ilustrado en la figura 4.1

Figura 4.1: Flujo de consulta de cliente

4.2.1. Archivo hooks.ts

El archivo hooks.ts es un archivo especial el cual se llama en cada consulta y antes del enrutamiento del servidor, esto es útil para la validación de datos (específicamente el control de acceso). Para lograr esto se entregan funciones que se detallan a continuación:

- `handle({event,response})`: Esta es la función que recibe cada consulta en el archivo, entrega como parámetros el argumento '`event`', el cual contiene información

de la consulta y la función '`response`' que llama al enrutador. Cabe destacar que en esta función se puede agregar información a la consulta para que posteriormente sea leída en los scripts del servidor. Para esto se llena el objeto '`event.locals`'

- `getSession(event)`: Función que se llama cada vez que se va a carga una página, los datos que se retornen de esta serán accesibles por el cliente lo que permite entregar datos de sesión básicos como el nombre del usuario actual.

4.2.2. Enrutador SvelteKit

El enrutador se encarga de enviar la consulta al lugar correspondiente, siendo este una página web o un script de servidor. Uno de los beneficios de este enrutador, es que permite el uso de parámetros dinámicos definidos entre [ ] los cuales pueden estar tanto en el nombre del archivo como en el nombre del directorio, esto permite que un mismo archivo o grupos de archivos puedan servir a diferentes consultas. Por otro lado, entrega distintas herramientas a las páginas y scripts que se detallaran a continuación.

4.2.3. Paginas o scripts de servidor

La lectura de la consulta en el último punto depende del tipo de archivo, en este punto nos enfocaremos únicamente de los scripts del servidor, quedando explicada la lectura en páginas en la sección 4.5

El procesamiento de la consulta por parte de los scripts se realiza mediante funciones que dependen del método de la consulta, estos pueden ser '`get()`' o '`post()`', los cuales entregan los siguientes argumentos:

- `request`: Datos de consulta realizada.
- `url`: Ruta solicitada por la consulta.
- `params`: Parámetros dinámicos obtenidos por el enrutador.
- `locals`: Datos locales definidos en procesamiento del archivo `hooks.ts`

Por otro lado, es necesario que estas funciones retornen un paquete HTTP[16] que sera entregado al cliente con los siguientes parámetros:

- `status`: código de estado HTTP
- `headers`: Cabecera del mensaje
- `body`: Contenido del mensaje

4.3. Interacción con plataformas externas

Con el fin de facilitar el desarrollo y escalabilidad del código se deciden implementar diversas librerías para ayudar en la interacciones con las distintas plataformas que se quieren interactuar, de estas se encuentran:

4.3.1. Pasaporte USM

La universidad presenta un sistema de cuentas que permite el acceso único a sus distintos servicios. Para poder interactuar con él se usa el protocolo LDAP[26], el cual permite verificar la información de usuario que se le entrega mediante el comando `bind`.

Para poder usar el comando `bind` es necesario tener la URL del servidor LDAP, como también la ruta de directorio en la cual se encuentran las cuentas de usuario a verificar. Con el fin de facilitar la interacción se usa el paquete `ldap-authentication`, la cual entrega la función `authenticate` con la que se puede verificar el usuario.

4.3.2. Crossref

Con el fin de facilitar la subida de publicaciones y cruzar la información con la ya existente, se implementa una librería que permite la interacción con la plataforma de cross-Ref, la cual permite hacer consultas a la API y reformatear la información obtenida para la lectura en los módulos. Las consultas se realizan bajo el método GET a la dirección `https://api.crossref.org/`. [6]

4.3.3. Base de datos

Para poder interactuar con la base de datos es necesario realizar una conexión a esta, para esto se usa el paquete '`mysql2`', el cual entrega funciones para realizar consultas y conectarse a esta. Por otro lado, para flexibilizar el código se crea una librería propia para traducir los datos de la aplicación a la base de datos y viceversa.

4.4. Modulos de backend

El desarrollo del backend se divide de manera resumida en distinto módulos como se muestra en la figura 4.2

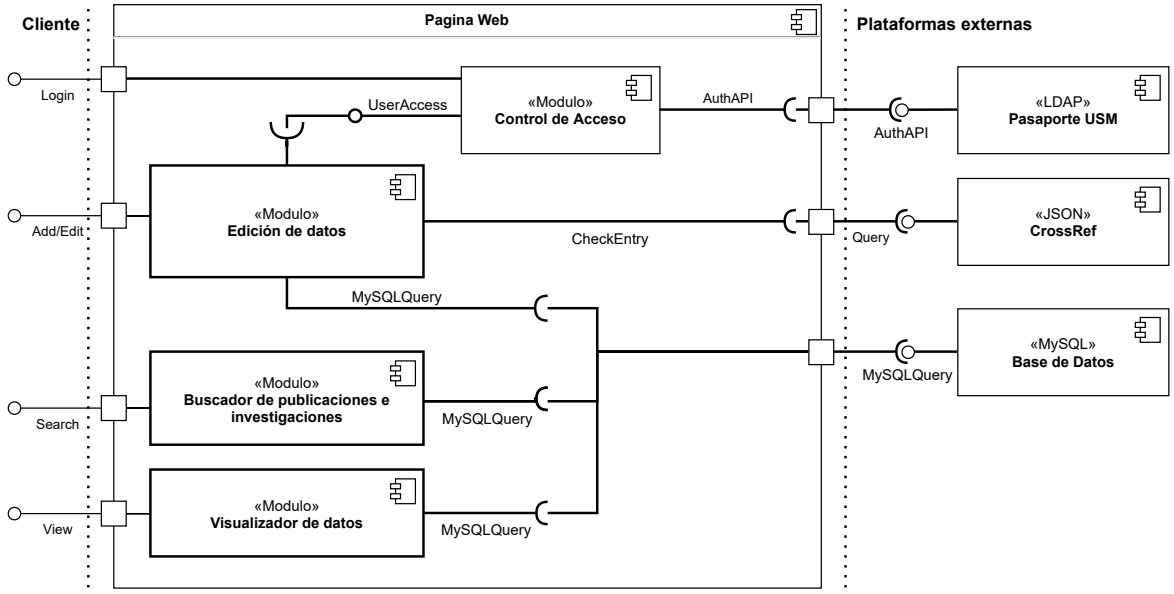


Figura 4.2: Modulos de backend

Cada módulo de la página recibe una interacción del cliente y realiza las consultas correspondientes a los distintos módulos, ya sean de la propia página o de plataformas externas (Que pueden estar en la misma máquina o en la web). En el caso de realizar consultas a plataformas externas, esto pasa previamente por una librería que traduce el mensaje de la página al código de la plataforma a consultar.

4.4.1. Control de acceso

Para controlar el acceso a los distintos módulos de la página se define un módulo de control de acceso, con el cual se permite que el usuario ingrese los datos de su cuenta USM, se pasen estos a la plataforma de pasaporte USM, y se le permite el acceso si tanto el pasaporte USM como la whitelist lo validaron.

Para mantener la sesión del usuario, se crea una cookie única para identificar el usuario y se guarda en la base de datos, lo cual permite mantener la sesión aun cuando la página se reinicie, todo esto siendo verificado cada vez que se realiza una consulta al servidor.

4.4.2. Edición de datos

Para editar la información de la base de datos se debe recibir una consulta con todos los datos necesarios para editar la información de la base de datos (los cuales llegan en formato JSON), para verificar la integridad de los datos recibidos se usa el paquete `jsonschema`, el cual entrega una función (`validate()`) donde se define la estructura de los datos y verifica la integridad de la consulta. Verificados los datos y el acceso se proceden a editar la base de datos, agregando o editando la información en la tabla correspondiente y sus tablas relacionadas.

Para facilitar la interacción del cliente se entregan distintas funciones de API descritas a continuación:

- `/api/db/getJournals.json`: Recibe una consulta GET con texto a buscar para obtener todas las revistas que lo contengan, lográndose mediante una consulta SQL a la base de datos:

```
SELECT id,title FROM journals WHERE title LIKE ? ORDER BY LOCATE(?,title)
```

- `/api/db/getAuthors.json`: Recibe una consulta GET con los nombres y apellidos del autor a buscar en la base de datos, consiguiéndose mediante la siguiente consulta SQL:

```
SELECT id,name,surname FROM authors WHERE name LIKE ? AND surname LIKE ? ;
```

- `/api/getPublicationFromDOI.json`: Recibe una consulta GET con el código DOI a buscar y devuelve los datos referentes a la publicación, para esto se realiza una consulta a la plataforma Crossref

- `/api/getAgencies.json`: Recibe una consulta GET con texto a buscar entre todas las agencias y devuelve las que se relacionen, esto se logra mediante la siguiente consulta SQL:

```
SELECT id,title,acronym FROM agencies WHERE title LIKE ? OR acronym LIKE ?
    ORDER BY LOCATE(?,acronym), LOCATE(?,title)
```

4.4.3. Buscador de publicaciones y proyectos

Para realizar el buscador de publicaciones y proyectos se toma una consulta que contiene un texto con el cual se consultan a las tablas de publications, authors, journals y projects para ver si contienen alguna publicación, autor, revista y/o proyecto con ese texto.

Para lograr obtener los datos se realiza una consulta a la base de datos con la sentencia **SELECT** que permite obtener los datos de una tabla o grupo de estas, las cuales se definen en la sentencia **FROM** donde se define la tabla principal y se unen las siguientes con la sentencia **LEFT JOIN**, uniendo en base a columnas de las tablas. Se agrupan los datos con **GROUP BY** en base a la id de la publicación y se restringen los datos con la sentencia **HAVING** donde se definen las condiciones. Cabe destacar que en la selección de datos se hace uso de la función `GROUP_CONCAT()`, que permite agrupar los datos de una tabla en un solo campo (por ejemplo, para agrupar los nombres de los autores de una publicación). Un ejemplo de esta consulta se encuentra en el anexo 6.

Por otro lado, para facilitar la interacción del cliente a la hora de querer realizar una búsqueda se entrega la función de API `/api/db/getActualAuthors.json` que entrega

los miembros del departamento actuales mediante una consulta SQL:

```
SELECT id,name,surname FROM authors WHERE usm='actual' ORDER BY surname;
```

4.4.4. Visualizador de datos

La obtención de datos a entregar se realiza mediante una consulta por ID del dato, obteniéndose del parámetro dinámico de la consulta, por ejemplo, al consultar por la ruta `/publications/534` se obtiene que el valor de ID es 534. Dicho esto, se consulta a la base de datos por esta respectiva ID en la tabla correspondiente (en el caso del ejemplo seria a la tabla publications), si el recurso no es encontrado se devuelve el error HTTP 404 y en caso de que si exista se devuelve toda la información relevante a este. Cabe destacar que en el caso que se consulte por revistas o autores se devuelve también todas las publicaciones (y proyectos en el caso de autores) relacionados a este.

4.5. Entrega de páginas

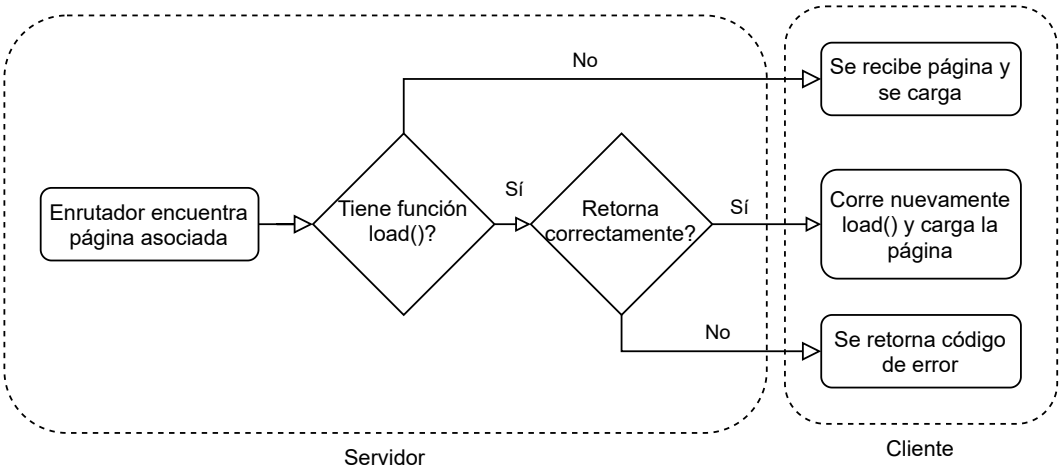


Figura 4.3: Flujo de carga de página para el cliente

Antes de hablar de la interfaz gráfica es necesario entender como el cliente obtiene la página y como se envían las consultas al servidor. Como se menciona en la sección 4.2, la página que se le muestra al cliente se le entrega después del enrutamiento en el servidor. En

el proceso de enrutamiento es posible correr código antes de entregar la pagina asociada, mediante la función `load()` definida en la misma página (Explicado en la figura 4.3). Cabe destacar que esta función corre en el servidor y en el cliente, por lo que no se le debe pasar información sensible directamente, para esto es mejor hacer una petición a un módulo del servidor por medio de la función `fetch()`. El formato de los archivos queda ilustrado en la figura 4.4

```
<script context="module">
    function load() {
        // Código ejecutado antes de cargar página
        // Corre en el servidor y en el cliente.
    }
</script>
-------------------------------------------------
<script>
    // Lógica de pagina en Javascript
</script>
-------------------------------------------------

<!-- Diseño de página en HTML -->

-------------------------------------------------
<style>
    /* Estilo de página en CSS */
</style>
```

Figura 4.4: Formato de archivo .svelte con svelteKit

4.6. Componentes

Como se explica en la sección de diseño (específicamente en la sección 3.4), se usan las siguientes componentes:

4.6.1. Componentes de gráficas

Para visualizar mejor la información se hace uso de gráficos, los cuales vienen incluidos en la librería de Chart.js, la que entrega diversas componentes que permiten crear gráficas de manera simple, visualmente atractivas y altamente personalizables, por lo que son perfectas para la interfaz.

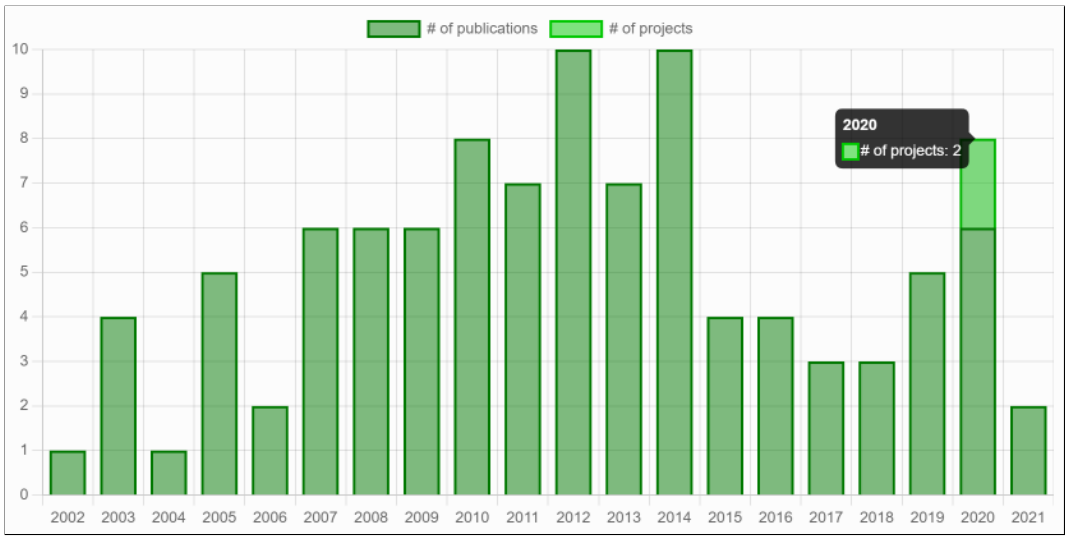


Figura 4.5: Gráfico de barras con la cantidad de publicaciones y proyectos por año.

4.6.2. Ingreso de autor y agencias

Componente '`Author`' (o `Agency`) que entrega campos para buscar a un autor por medio de una consulta GET a '`/api/db/getAuthors.json`' o agencias por consulta a '`/api/db/getAgencies.json`', lo cual retorna una tabla con sugerencias de autores o agencias relacionados para presentarlos en el autocompletado, esto permite rellenar todos los campos por medio de una búsqueda por los campos.

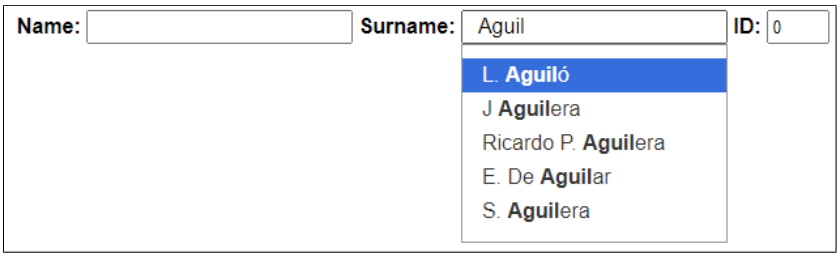


Figura 4.6: Componente para el ingreso de autor.

4.6.3. Parámetros de búsqueda

Componente que contiene los parámetros de búsqueda, los cuales afectan a la muestra de información en el componente de publicación, cabe destacar que el campo *Department members* aparece solo si se le pasan los datos (En este caso, solo en las búsquedas de la

Figura 4.7: Componente con los parámetros de búsqueda.

página de inicio), y además permite el ingreso de más de un autor gracias al componente '*svelte-multiselect*'

4.6.4. Contenedor de publicaciones

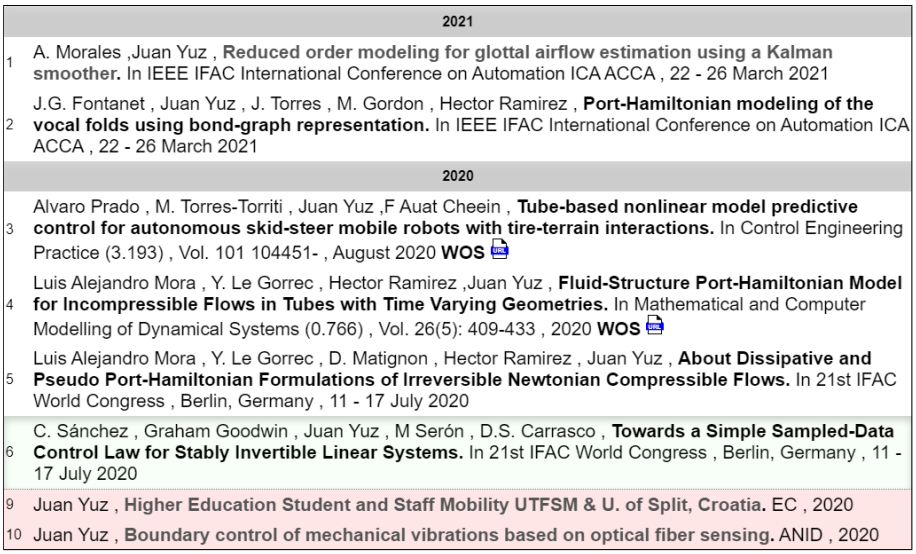


Figura 4.8: Componente para mostrar las publicaciones.

Componente '`Publications`' que muestra las publicaciones y proyectos en una lista en formato BiBTeX, siendo estos datos ordenados según las opciones de la componente '`searchOptions`', por otro lado, los datos también se vuelven a ordenar si se actualiza la tabla original. Cabe destacar que los proyectos se muestran con un rojo claro, mientras que la publicación a seleccionar con un verde claro.

4.7. Desarrollo de páginas

4.7.1. Barra de navegación

Figura 4.9: Barra de navegación

Para realizar la barra de navegación se usa el archivo ’__layout . svelte ’, el cual asegura que su contenido este presente en todas las páginas, con esto se ideo una barra simple como se aprecia en la figura 4.9, la cual contiene el acceso directo a la página de inicio sesión y a la página de inicio.

Figura 4.10: Barra de navegación con sesión iniciada.

La barra de navegación muestra su contenido dependiendo de si el usuario tiene iniciada la sesión o no. Para esto se revisa la variable `session` para ver si ha recibido cambio alguno (El cual se puede realizar en la función `getSession()` del archivo `hooks.ts`).

4.7.2. Ingreso de sesión

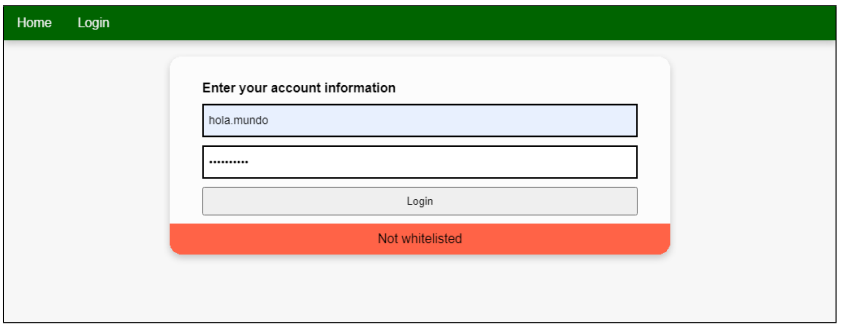


Figura 4.11: Página de inicio de sesión

Para iniciar sesión se diseña una caja simple con campos para el ingreso de usuario y clave, junto al botón de ingreso. Por otro lado, se deja un campo en la parte inferior para mostrar todo error que pueda ocurrir. Como se ilustra en la figura 4.11

4.7.3. Inicio

Figura 4.12: Pagina inicio

Para diseñar la página de inicio se modifica el archivo '`index.svelte`', el cual presenta el contenido al ingresar a la interfaz, siendo este un simple buscador con filtros que permite buscar por miembros actuales del departamento, por tipo y/o por fecha. Por otra parte, permite elegir el orden en que se muestra el contenido y la agrupación de este. Los miembros actuales del departamento se obtienen previo a la carga de la página llamando a '`getActualAuthors.json`' del servidor.

Para obtener las publicaciones relacionadas con la búsqueda se realiza una consulta GET a '`/api/db/search`', el cual retorna una tabla con todas las publicaciones relacionadas. Estas publicaciones se pasan a la componente '`Publications`', la cual muestra y ordena todas las publicaciones. Como se muestra en la figura 4.13

4.7.4. Visualizador de datos

Para mostrar la información se consulta al servidor a una ruta que depende de la información que se solicita, para esto se usa la ID y el tipo de dato a pedir. Por otro lado, según el tipo de dato se muestra o no las publicaciones y/o proyectos relacionados a este.

En el caso de autores, revistas y proyectos se muestran sus publicaciones y/o revistas relacionadas

Figura 4.13: Página inicio con publicaciones

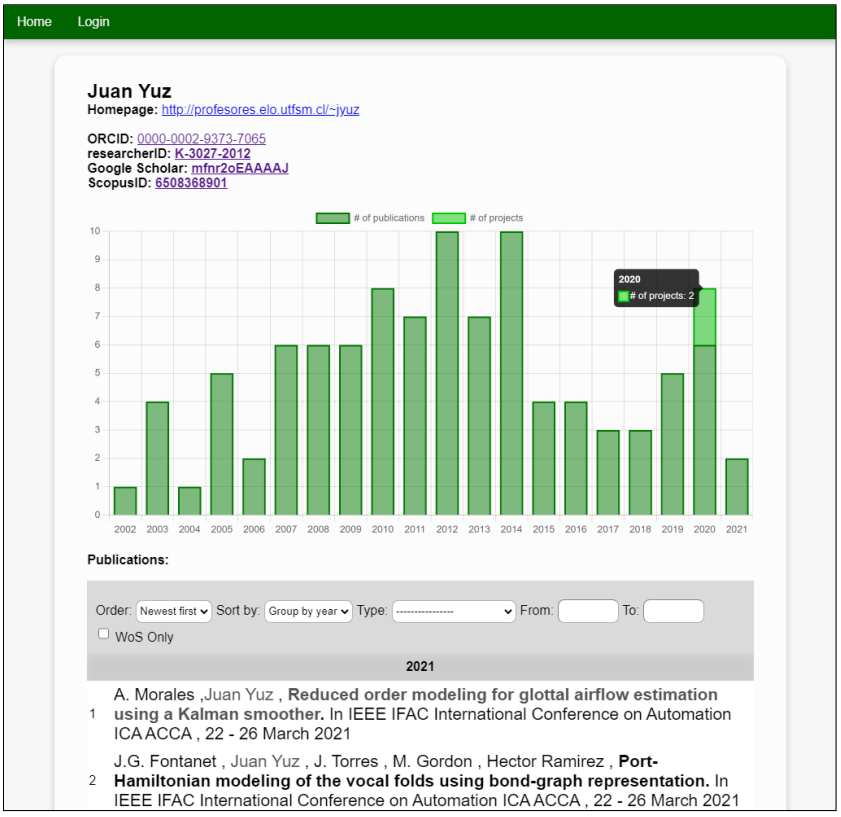


Figura 4.14: Página de información de autor

Capítulo 5

Despliegue de interfaz

En el presente capitulo se describen todos los pasos necesarios para el despliegue de la interfaz y las pruebas realizadas a esta. Para esto, la interfaz se monta en una máquina virtual con sistema operativo centOS 7.9, el cual requiere de los siguientes pasos.

5.1. Instalación y configuración de paquetes

Para que la interfaz pueda funcionar sin problemas es necesario instalar y configurar los paquetes que a continuación se detallan:

5.1.1. Node Version Manager

Node Version Manager (NVM) es un paquete que simplifica la instalación y configuración de Node.js, para instalar se usan los siguientes comandos bash:

```
1 #!/bin/bash
2 curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh |
    bash
3 nvm install --lts
```

Listing 5.1: Comandos de instalación de NVM

Estos comandos descargan e instalan NVM para posteriormente instalar la última versión de producción de Node.js

5.1.2. Nginx

Nginx es un servidor proxy inverso, el cual recibe consultas de la web y las redirige al servidor web interno (Node.js en este caso). Esto permite que los usuarios que ingresen a la dirección web tanto por protocolo HTTP o por HTTPS sean redirigidos al servidor web sin problemas.

Para la instalación se usa la propia aplicación de paquetes de centOS (yum) y se configuran los archivos para permitir la funcionalidad buscada.

```
1 sudo yum install nginx
2 sudo setsebool -P httpd_can_network_connect 1
3 mkdir /etc/nginx/sites-available
4 mkdir /etc/nginx/sites-enabled
5 sudo mkdir /etc/nginx/ssl
```

Listing 5.2: Comandos de instalación de nginx

Dado el firewall configurado en la máquina virtual es necesario el uso del segundo comando para permitir la redirección de consultas. Por otro lado, se crean las carpetas para dejar los archivos de configuraciones los cuales están descritos en los anexos 6.1 y 6.2. Cabe destacar que, para cargar configuraciones de sitios, estos deben estar en la carpeta '`sites-enabled`', como además los certificados SSL se deben ubicar en su carpeta respectiva.

5.1.3. MySQL

Para instalar y configurar el servidor MySQL se usan los siguientes comandos, los cuales modifican el repositorio a consultar e instalan la versión 8.0:

```
1 sudo yum-config-manager --disable mysql57-community
2 sudo yum-config-manager --enable mysql80-community
3 sudo yum install mysql-server
4 sudo service mysqld start
```

Listing 5.3: Comandos de instalación de MySQL

Es importante prestar atención a los mensajes de instalación, ya que en estos se mencionan los pasos para ingresar con acceso root a la base de datos. Con esto es posible configurar los usuarios para el acceso de la interfaz, como también el acceso externo. Para agregar nuevos usuarios se usa el siguiente comando MySQL, el cual crea un usuario con su respectiva contraseña con ingreso de la IP designada:

```
CREATE USER 'usuario'@'IP'
  IDENTIFIED BY 'password';
GRANT ALL
  ON *.*
  TO 'usuario'@'IP'
  WITH GRANT OPTION;
```

Listing 5.4: Comandos para la creación de usuarios en la base de datos

5.1.4. Firewall

La máquina virtual viene con un firewall incluido, el cual es necesario configurar para permitir el acceso por los puertos que usan las aplicaciones instaladas, para esto se usan los siguientes comandos:

```
1 sudo firewall-cmd --zone=public --add-port=80/tcp
2 sudo firewall-cmd --zone=public --add-port=443/tcp
3 sudo firewall-cmd --zone=public --add-port=3000/tcp
```

Listing 5.5: Comandos para abrir puertos

Los puertos 80 y 443 son de los protocolos HTTP y HTTPS respectivamente, mientras que el puerto 3000 es el que usa la interfaz desarrollada en Node.js

5.1.5. PM2

PM2 es una aplicación que facilita el monitoreo de los servidores Node.js como también permite subir automáticamente los servidores en caso de caerse (Su interfaz se muestra en la figura 5.1). Para instalarlo simplemente se hace uso del siguiente comando bash

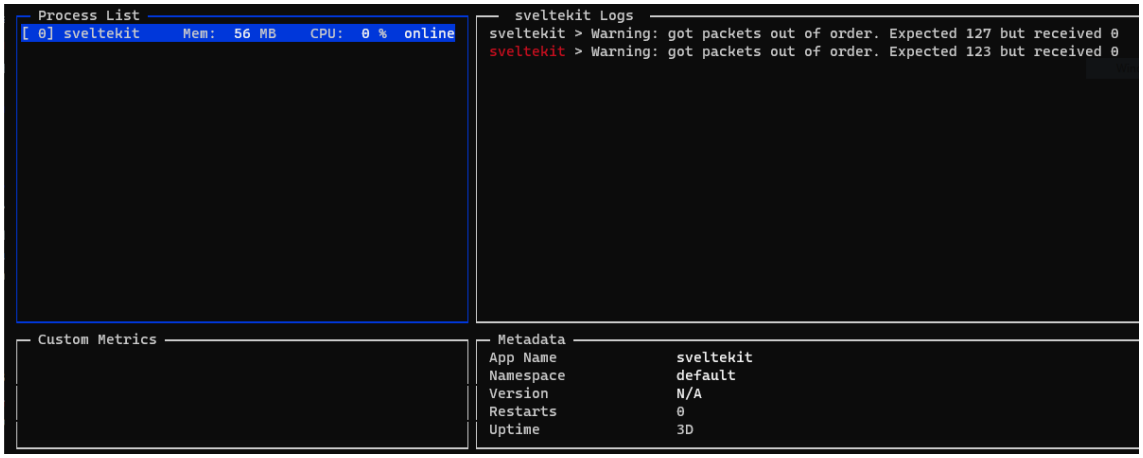


Figura 5.1: Consola de monitoreo de PM2

```
1 npm install pm2 -g
```

Listing 5.6: Comando de instalación de PM2

5.2. Migración de base de datos

Para migrar la base de datos de la interfaz de '`altair.elo.utfsm.cl`' es necesario primero crear el esquema en el cual se va a trabajar, para esto se opto por usar MySQL Workbench, que permite crear y migrar bases de datos fácilmente.

Para empezar se usa un modelo de la base de datos, con el cual se puede crear el esquema en el servidor remoto. En el programa se abre el archivo '`modelo.mwb`' el cual contiene el modelo y se sincroniza ingresando a '`Database > Synchronize model`', el cual abre un asistente donde se ingresan los datos de conexión y se elige los campos a sincronizar.

Una vez sincronizado el modelo, se tiene el esquema 'BIBELO' que es el que usa la interfaz desarrollada en esta memoria, mientras que el esquema 'pub_ELO' pertenece a la plataforma existente en '`altair.elo.utfsm.cl`', por lo que se requiere transferir los datos usando los comandos MySQL que se encuentran en el archivo ' despliegue . sql' (Anexo A4), el cual se ejecuta con el siguiente comando:

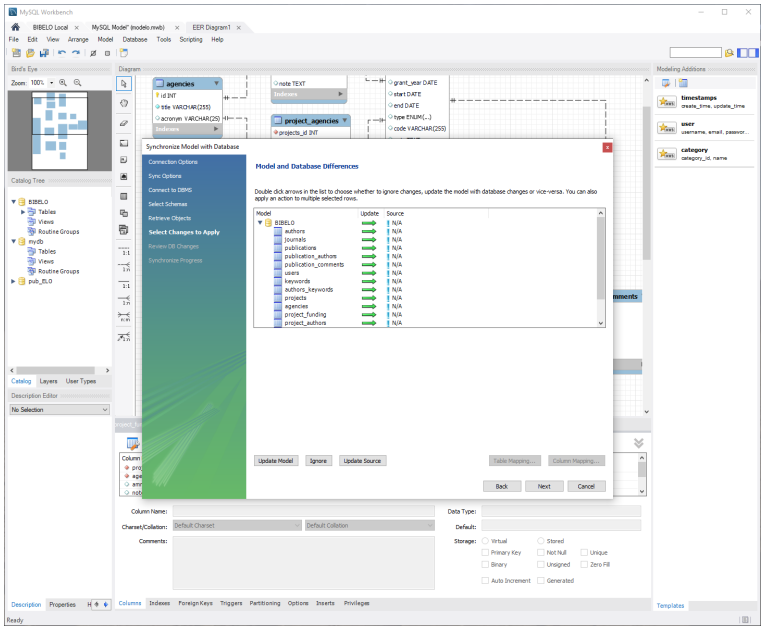


Figura 5.2: MySQL Workbench

```
mysql -u <usuario> -p <schema> < despliegue.sql
```

Listing 5.7: Comando para ejecutar archivo despliegue.sql

5.2.1. Respaldo periódico de base de datos

Para generar un respaldo de la base de datos se usan los siguientes comandos.

```
!/bin/bash
_file="backup_$(date +"%d-%m-%Y").sql"
mysql -u 'root' -p<clave> BIBELO > "$_file"
```

Listing 5.8: Archivo bash (dumpsql.sh) para respaldar base de datos

Cabe destacar que para ejecutar los comandos es necesario usar 'chmod +x <archivo>'.

Para lograr que el respaldo sea automático se usa la aplicación *Cron* de UNIX, la cual permite ejecutar archivos cada cierto tiempo, lo cual se logra agregando el archivo mediante 'crontab —e', el cual abre un editor donde se agrega la siguiente línea:

```
0 0 1 * * ~/scripts/dumpsql.sh # Take backup every month
```

5.3. Despliegue de la interfaz y pruebas

Para el despliegue de la interfaz en el servidor del departamento se hace uso del software GIT, que permite mantener un control de las versiones de los archivos y permite subir y obtener fácilmente los archivos de un repositorio. Para esto se hace uso del comando '`git clone`' el cual permite clonar el repositorio en el servidor del departamento. Una vez obtenidos los archivos se procede a definir las variables de la aplicación en el archivo '`.env`', el cual contiene los datos de acceso a la base de datos y la contraseña que usara el administrador. Por otro lado, las cuentas USM a las que se les van a permitir el acceso en el archivo '`.whitelist`'.

Una vez que se tenga la interfaz ya configurada se procede a levantar con los siguientes comandos:

```
npm run build
npm run production
```

Listing 5.9: Comandos para compilar y correr la interfaz

Estos comandos compilan la aplicación para que se pueda correr en Node.js y posteriormente se deja corriendo el servidor. Los comandos build y production se encuentran definidos en el archivo '`package.json`'

Para las pruebas se evalúa el funcionamiento correcto de las funciones e integraciones de la interfaz, como también el consumo de esta en su funcionamiento. De esto se pueden rescatar ciertos puntos:

- Inestabilidad de consulta por DOI a Crossref: Durante las pruebas de la interfaz se vio que ciertas publicaciones no entregan toda la información que deberían tener. Esto puede deberse a información incompleta en la base de datos de Crossref.

- Consumo fuera de lo normal en consultas de búsqueda: Si bien no es perceptible la demora a la hora de entregar la información, al analizar la consulta SQL, es posible observar que toma todos los datos de la tabla de publicaciones y revisa sobre todos ellos cada cláusula que se le entrega, esto se debe al uso de '`HAVING`' en vez de '`WHERE`', el cual se usa debido a que permite analizar frente a agrupaciones de datos ('`GROUP_CONCAT`'), cosa que no se puede hacer si se usa la cláusula '`WHERE`'

Capítulo 6

Conclusiones

En esta memoria de titulación se entregaron los procesos de diseño, desarrollo y despliegue del software de una nueva interfaz de publicaciones y proyectos para el Departamento de Electrónica. Con respecto a los objetivos iniciales planteados se establece lo siguiente:

1. Con el fin de visualizar la productividad individual se entrega la capacidad de observar la cantidad de publicaciones y proyectos realizados al año, como también se entregan estos en formato trasladar a un curriculum. Por otro lado, también permite buscar por varios autores a la vez, permitiendo mirar publicaciones en conjunto (AND) y/o, alternativamente, obtener las publicaciones en que algún miembro del grupo se encuentre (OR).

2. Es posible consultar la base de datos por medio de la página de búsqueda inicial, que entrega diferentes filtros y busca en las tablas de publicaciones, autores, proyectos, revistas y agencias, como también se permite consultar por alguna entrada determinada directamente desde la URL con la ID que tiene en la base de datos.

3. Es posible cargar información de las publicaciones por medio del DOI, el cual realiza las consultas a la plataforma de Crossref, facilitando el ingreso de nuevas publicaciones y su edición.

4. Es posible incorporar proyectos de investigación y desarrollo a la base de datos, permitiendo el ingreso de los miembros, las fechas de inicio y termino, las agencias

relacionadas y el financiamiento que tengan. Por otro lado, estos datos se muestran en el perfil de un miembro o en la búsqueda grupal.

Dicho lo anterior, se establece que se cumplieron los objetivos, pero aún hay espacio para mejorar, lo cual nos lleva al trabajo futuro de la interfaz, el cual da lugar a las siguientes mejoras que se pueden realizar:

- Aumentar la integración con plataformas externas: Si bien ya se tiene una integración para el apartado de publicaciones, se espera poder mejorar la integración para poder ingresar publicaciones en bloque y por otro lado integrar con agencias locales (Por ej. ANID) y externas (como Google Schoolar) para facilitar el ingreso de proyectos a la base de datos.

- Mejorar el apartado visual de la interfaz: Actualmente la interfaz carece de gráficas y animaciones que mejoren la intuición e interacción con el usuario, por lo que si se quiere llevar a un producto comercial es necesario mejorar este aspecto para que sea más atractivo a los usuarios.

- Mejorar el aspecto de las cuentas en la interfaz: En un principio se buscaba permitir el acceso de todos los usuarios USM asociados a un autor, limitados a modificar sus publicaciones y página de autor. Esto no se pudo llevar a cabo en el plazo establecido, estableciendo un ingreso según lista con acceso y control total (Excepto a la lista).

En conclusión, se entrega una interfaz que entrega mayores funcionalidades que la anterior, permite la visualización de los proyectos y se integra a otras plataformas para facilitar su uso. Se espera que esta interfaz sea de gran utilidad para el departamento y se seguirá trabajando en los puntos que se pueden mejorar.

Bibliografía

[1] *Angular*. [en línea] <https://angular.io/features> [consulta: 01-06-2021].

[2] Adam Barth. *HTTP State Management Mechanism*. RFC 6265. Abr. de 2011. [en línea] <https://www.rfc-editor.org/info/rfc6265>.

[3] T. Berners-Lee, R. Fielding y L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Ene. de 2005. [en línea] <https://www.rfc-editor.org/info/rfc3986>.

[4] *C#*. [en línea] <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/> [consulta: 01-06-2021].

[5] *Digital Object Identifier*. [en línea] <https://www.doi.org/> [consulta: 17-01-2022].

[6] *Documentación de API de Crossref*. [en línea] <https://api.crossref.org/swagger-ui/index.html> [consulta: 21-01-2022].

[7] *DSpace*. [en línea] <https://duraspace.org/dspace/about/> [consulta: 11-05-2021].

[8] *Especificación CSS*. [en línea] <https://www.w3.org/TR/CSS/#css> [consulta: 23-01-2022].

[9] *Go*. [en línea] <https://go.dev> [consulta: 01-06-2021].

[10] *Google Scholar*. [en línea] <https://scholar.google.com/intl/en/scholar/about.html> [consulta: 11-05-2021].

[11] *ISSN*. [en línea] <https://www.issn.org> [consulta: 23-01-2022].

[12] *Java*. [en línea] <https://www.java.com/es/about/whatis_java.jsp> [consulta: 01-06-2021].

[13] *Mendeley*. [en línea] <https://www.elsevier.com/solutions/mendeley> [consulta: 24-05-2021].

[14] *MongoDB*. [en línea] <https://www.mongodb.com/es> [consulta: 01-06-2021].

[15] *MySQL*. [en línea] <https://www.mysql.com> [consulta: 01-06-2021].

[16] Henrik Nielsen y col. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. Jun. de 1999. [en línea] <https://www.rfc-editor.org/info/rfc2616>.

[17] *Node.js*. [en línea] <https://nodejs.org/es/about/> [consulta: 01-06-2021].

[18] *ORCID*. [en línea] <https://info.orcid.org/what-is-orcid/> [consulta: 24-05-2021].

[19] *PHP*. [en línea] <https://www.php.net/manual/es/intro-whatcando.php> [consulta: 01-06-2021].

[20] *PostgreSQL*. [en línea] <https://www.postgresql.org> [consulta: 01-06-2021].

[21] *Publons*. [en línea] <https://publons.com/about/home> [consulta: 24-05-2021].

[22] *Pure*. [en línea] <https://www.elsevier.com/solutions/pure/rims> [consulta: 11-05-2021].

[23] *Python*. [en línea] <https://www.python.org> [consulta: 01-06-2021].

[24] *React*. [en línea] <https://es.reactjs.org> [consulta: 01-06-2021].

[25] *Scopus*. [en línea] <https://service.elsevier.com/app/answers/detail/a_id/15100/supporthub/scopus/> [consulta: 24-05-2021].

[26] Jim Sermersheim. *Lightweight Directory Access Protocol (LDAP): The Protocol*. RFC 4511. Jun. de 2006. [en línea] <https://www.rfc-editor.org/info/rfc4511>.

[27] *Svelte*. [en línea] <https://svelte.dev> [consulta: 01-04-2022].

[28] *VIVO*. [en línea] <https://duraspace.org/vivo/about/> [consulta: 11-05-2021].

[29] *Vue*. [en línea] <https://vuejs.org> [consulta: 01-06-2021].

[30] *Web Of Science*. [en línea] <https://clarivate.com/webofsciencegroup/solutions/web-of-science/> [consulta: 17-01-2022].

Anexo A1

```
user nginx;
worker_processes auto;
pid /run/nginx.pid;
include /usr/share/nginx/modules/*.conf;
events {
        worker_connections 768;
}
http {
        access_log  /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;
        sendfile            on;
        tcp_nopush          on;
        tcp_nodelay         on;
        keepalive_timeout   65;
        types_hash_max_size 4096;
        include             /etc/nginx/mime.types;
        default_type        application/octet-stream;
        include /etc/nginx/conf.d/*.conf;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;
        include /etc/nginx/sites-enabled/*;
}
```

Listing 6.1: /etc/nginx/nginx.conf

Anexo A2

```
server {
        listen 80;
        listen [::]:80;
        #SSL
        listen 443 ssl;
        listen [::]:443 ssl;
        ssl_certificate         /etc/nginx/ssl/cert.pem;
        ssl_certificate_key     /etc/nginx/ssl/key.pem;

        server_name bibelo;

        if ($scheme != "https") {
                return 301 https://$host$request_uri;
        }

        location / {
                proxy_set_header        Host $host;
                proxy_set_header        X-Real-IP $remote_addr;
                proxy_set_header        X-Forwarded-For
                   $proxy_add_x_forwarded_for;
                proxy_set_header        X-Forwarded-Proto $scheme;

                proxy_pass              http://127.0.0.1:3000;
                proxy_read_timeout      90;

                proxy_redirect          off;
        }
}
```

Listing 6.2: /etc/nginx/sites-available/bibelo

Anexo A3: Búsqueda en base de datos

```
SELECT
        publications.id AS pubid,
        publications.title,
        publications.entry_type AS type,
        publications.volume AS volume,
        publications.number AS number,
        publications.pages AS pages, publications.booktitle AS booktitle,
        publications.location AS location,
        publications.year AS year,
        publications.month1 AS month1,
        publications.month2 AS month2,
        publications.day1 AS day1,
        publications.day2 AS day2,
        publications.url AS url,
        publications.pdf AS pdf,
        GROUP_CONCAT(authors.id) AS authid,
        GROUP_CONCAT(CONCAT(authors.name, " ",authors.surname) SEPARATOR "; ") AS
            authname,
        GROUP_CONCAT(publication_authors.place) AS authplace,
        journals.id AS journal_id,
        journals.title AS journal,
        journals.impact_factor AS impact_factor,
        journals.wos AS wos
FROM publications
        LEFT JOIN publication_authors ON publications.id = publication_authors.
            publications_id
        LEFT JOIN authors ON publication_authors.authors_id = authors.id
        LEFT JOIN journals ON publications.id_journal = journals.id
GROUP BY publications.id
HAVING
        (publications.title LIKE '%busqueda%'
        OR journals.title LIKE '%busqueda%'
        OR LOCATE('busqueda',authname)>0);
```

Anexo A4: despliegue.sql

```
INSERT INTO BIBELO.journals (id,title,issnp,issne,impact_factor,wos)
        SELECT id,title,issn_p,issn_e,ifactor,wos FROM pub_ELO.bib_journals;
INSERT INTO BIBELO.publications ( id, title, entry_type, uploaded, id_journal,
    year, month1, day1, month2, day2, booktitle, volume, number, pages, location,
    url, url_access, pdf, pdf_access, note, nousm, toappear, national_conference )
        SELECT id,title,entry_type,date_c,jid,year,month1,day1,month2,day2,
            booktitle,volume,number_,pages,address,url,IF(acc_url = '1',b'1',b'0')
            ,pdf,IF(acc_pdf = '1',b'1',b'0'),note,IF(nousm = 1,b'1',b'0'),IF(
            toappear = 1,b'1',b'0'),IF(cn = 1,b'1',b'0') FROM pub_ELO.
            bib_publications;
INSERT INTO BIBELO.authors (id,surname,name,usm,homepage)
        SELECT id,name,surname,IF(indexed = 4,'no',indexed+0),homepage FROM
            pub_ELO.bib_authors;
INSERT INTO BIBELO.publication_authors (publications_id,authors_id,type,place)
        SELECT id_pub,id_author,IF(tip='a','author','editor'),ordine FROM pub_ELO.
            bib_pub_to_auth
        WHERE EXISTS (SELECT id FROM BIBELO.publications WHERE id =
            bib_pub_to_auth.id_pub);
INSERT INTO BIBELO.users (user) VALUES ('admin');
```

Listing 6.3: despliegue.sql